МИНОБРНАУКИ РОСИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Информационная безопасность систем и технологий»

Курсовая работа
по дисциплине «Технологии и методы программирования»
на тему «Система проверки лицензии программы онлайн»
ПГУ.100503.С1115.КР.18ПИ118.01.ПЗ

Специальность – 10.05.03 Информационная безопасность автоматизированных систем
Специализация – Защищенные автоматизированные системы управления

Выполнил студент : _____ Новиков Д.О.
Группа: 18ПИ1
Руководитель:
_____ Лупанов М.Ю.
Работа защищена с оценкой _____
Дата защиты 13.06.19

2019

Реферат

Отчет о курсовой работе содержит 32 страниц, 3 таблицы, 6 рисунков, 4 источника, 4 приложения.

VISUAL STUDIO, C++, КЛАСС, МЕТОД, ПАРАМЕТРЫ, ФУНКЦИЯ, ДИАГРАММА, МОДУЛЬ, DOXYGEN, ИМЯ ПОЛЬЗОВАТЕЛЯ, КЛИЕНТ, СЕРВЕР WINDOWS.

Объектом курсовой работы является изучение и практическое применение программных средств Microsoft Visual Studio.

Цель курсовой работы — разработка модуля проверки лицензионного ключа программы в режиме онлайн, а так же документирование проекта с использованием пакета Doxygen.

В ходе выполнения курсовой работы была изучена работа с многомодульными проектами, строками, клиент-серверным взаимодействием.

В результате выполнения курсовой работы был создан модуль проверки лицензии программы онлайн.

«УТВЕРЖДАЮ»
Зав. кафедрой ИБСТ
Зефиров С.Л.
16 05 2019.

Задание
на курсовую работу

по теме: Система проверки лицензии программы онлайн

1 Дисциплина: Технологии и методы программирования.

2 Студент: Новиков Д.О.

3 Группа: 18ПИ1.

4 Исходные данные на работу:

4.1 Цель: разработка модуля для проверки лицензии программы с выходом в сеть Интернет .

4.2 Требования к модулю:

- модуль должен обеспечивать выполнение операции проверки алгоритма создания ключа на подлинность;
- модуль должен обращаться к серверу для проверки лицензии;
- тип интерфейса, используемого в программе, – командная строка;
- интерфейс программы должен содержать:
  - имя пользователя;
  - лицензионный ключ;
  - результат проверки;
- модуль должен функционировать в операционной системе Windows 10;
- Передача данных осуществляется в виде объектов JSON поверх протокола TCP/IP.
- Внутри хранится лицензионный ключ и имя пользователя.
- программа должна быть документирована с использованием пакета документирования Doxygen.

5 Структура работы

5.1 Пояснительная записка(содержание работы):

- словесная модель программы ;
- диаграмма вариантов использования и диаграмма классов ;
- диаграмма последовательностей и диаграмма деятельности ;
- разработка модуля ;

- разработка модульных тестов ;
- разработка функциональных тестов ;

5.2 Экспериментальная часть:

- обнаружение дефектов в модуле с помощью модульного тестирования
- проверка модуля на соответствие техническому заданию с помощью квалификационного тестирования

6 Календарный план выполнения курсовой работы

- оформление технического задания к 6 марта 2019;
- разработка диаграмм вариантов использования и диаграмм классов к 20 марта 2019;
- разработка диаграмм последовательностей и диаграмм деятельности к 3 апреля 2019;
- разработка кода модуля к 17 апреля 2019;
- разработка модульных тестов и выполнение тестирования к 1 мая 2019;
- разработка функциональных тестов и приемочное тестирование к 15 мая 2019;
- построение диаграмм к 29 мая 2019;
- разработка документации к 29 мая 2019;
- оформление отчета о курсовой работе к 29 мая 2019;

Дата защиты работы « » 2019г.

Руководитель работы _____М.Ю. Лупанов

Задание получил « » 2019г.

Студент _____Д.О. Новиков

Нормоконтролер _____М.Ю. Лупанов

Содержание

Введение.....6
1 Пояснительная записка.....7
1.1 Словесная модель программы.....7
1.2 Разработка диаграмм вариантов использования и классов.....7
1.3 Разработка диаграмм последовательностей и деятельности 9
1.4 Разработка программного модуля.....12
1.5 Разработка модульных тестов.....13
1.6 Разработка функциональных тестов.....13
1.7 Документирование программы с использованием Doxygen.....14
2 Экспериментальная часть.....15
2.1 Обнаружение дефектов в модуле с помощью модульного тестирования.....15
2.2 Проверка модуля на соответствие техническому заданию с помощью квалифицированного тестирования.....16
Заключение.....17
Список используемых источников.....18
Приложение А.....19
Приложение Б.....21
Приложение В.....29
Приложение Г.....31
Приложение Д.....32

Введение

Лицензия на программное обеспечение является юридическим документом, регулирующим использование и распространение программного обеспечения, охраняемого авторским правом. Все программное обеспечение, не находящиеся в свободном доступе, защищены авторским правом. Типичная лицензия дает конечному пользователю разрешение на использование одного или более экземпляров программного обеспечения в работе, не влечет за собой нарушение прав издателя, в соответствии с законом об авторском праве. По сути, лицензия действует как обещание от издателей программного обеспечения, не подать в суд на конечного пользователя, пользующегося правами, принадлежащими издателю программного обеспечения. Лицензирование программного обеспечения, развиваясь, меняет представление о том, как организации и индивидуальные пользователи приобретают и применяют программы. Для компаний используемая ими модель лицензирования может решающим образом повлиять на прибыль. Для этого и используется система проверки лицензии[1].

Объектом исследования курсовой работы является система проверки лицензионного ключа.

Целью курсовой работы является выработка практических навыков разработки многомодульных проектов на языке C++.

Задачей работы является разработка модуля проверки лицензии программы в режиме онлайн.

Данная задача была решена с помощью программного продукта Microsoft Visual Studio.

1 Пояснительная записка

1.1 Словесная модель программы

Модуль проверки лицензии в режиме онлайн содержит в себе реализацию клиент серверного взаимодействия, проведение валидации.

Для успешной разработки модуля проверки лицензии и выполнения заданий по курсовой работе необходимо было определиться с используемой библиотекой для обеспечения интернет взаимодействия, при помощи которого данные клиента будут отправляться на сервер, обрабатываться на нем и возвращаться обратно. Для разработки данного модуля был использована библиотека Windows Sockets API.

В качестве формата данных было решено использовать объекты Json.

Для валидации объекта Json были использованы технологии регулярных выражений встроенные в стандартную библиотеку C++.

Для удобства использования модули были реализованы в виде абстрактных классов.

Функция handle получает результат обработки данных сервером и обрабатывает их в зависимости от реализации пользователя.

Функция execute была использована для отправки клиентом запроса на сервер.

Функции ok bad deny возвращают строки которые соответствуют вердикту сервера: ok при присутствии лицензионного ключа, deny при его отсутствии, bad при неправильно составленном запросе.

Функция fillDB была использована для того что бы заполнить базу данных сервера.

1.2 Разработка диаграмм вариантов использования и классов

Диаграмма вариантов использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой.

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им

действительно необходимы некоторые варианты использования. [3]

Диаграмма вариантов использования для модуля приведена на рисунке 1.

Диаграмма классов — это структурная диаграмма, на которой показано множество классов, интерфейсов, кооперацией и отношений между ними. В UML диаграмма классов является типом диаграммы статической структуры. Она описывает структуру системы, показывая её классы, их атрибуты и операторы, и также взаимосвязи этих классов.

Диаграмма классов UML - это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами - отношения между узлами (ассоциации, наследование, зависимости).[3]

Диаграмма классов приведена на рисунке 2.

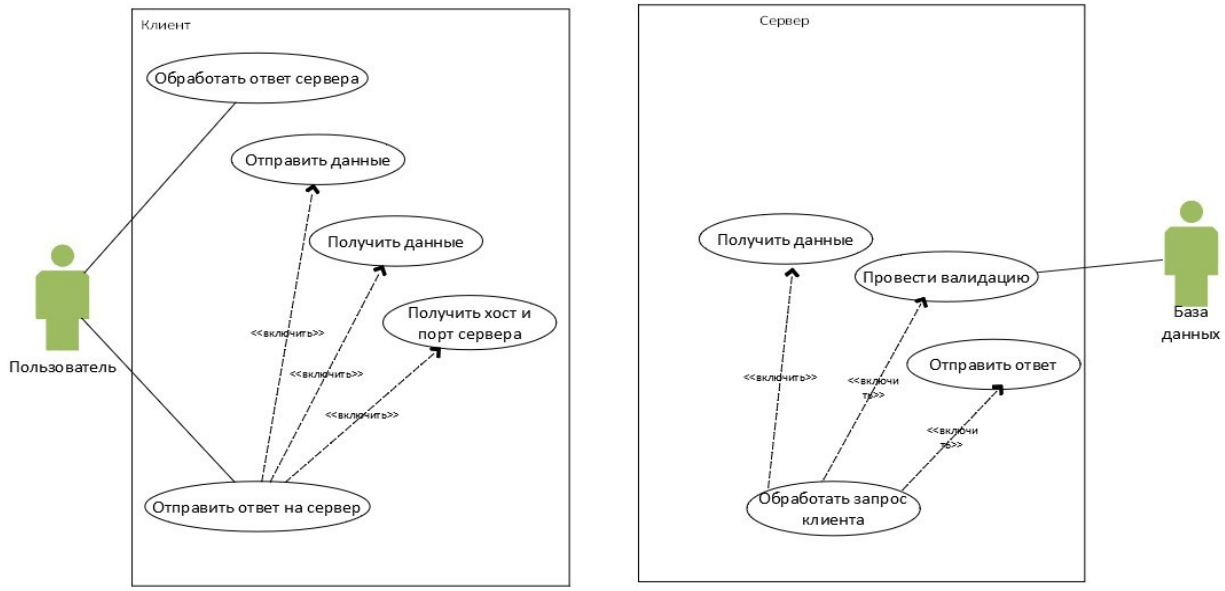


Рисунок 1 - Диаграмма вариантов использования

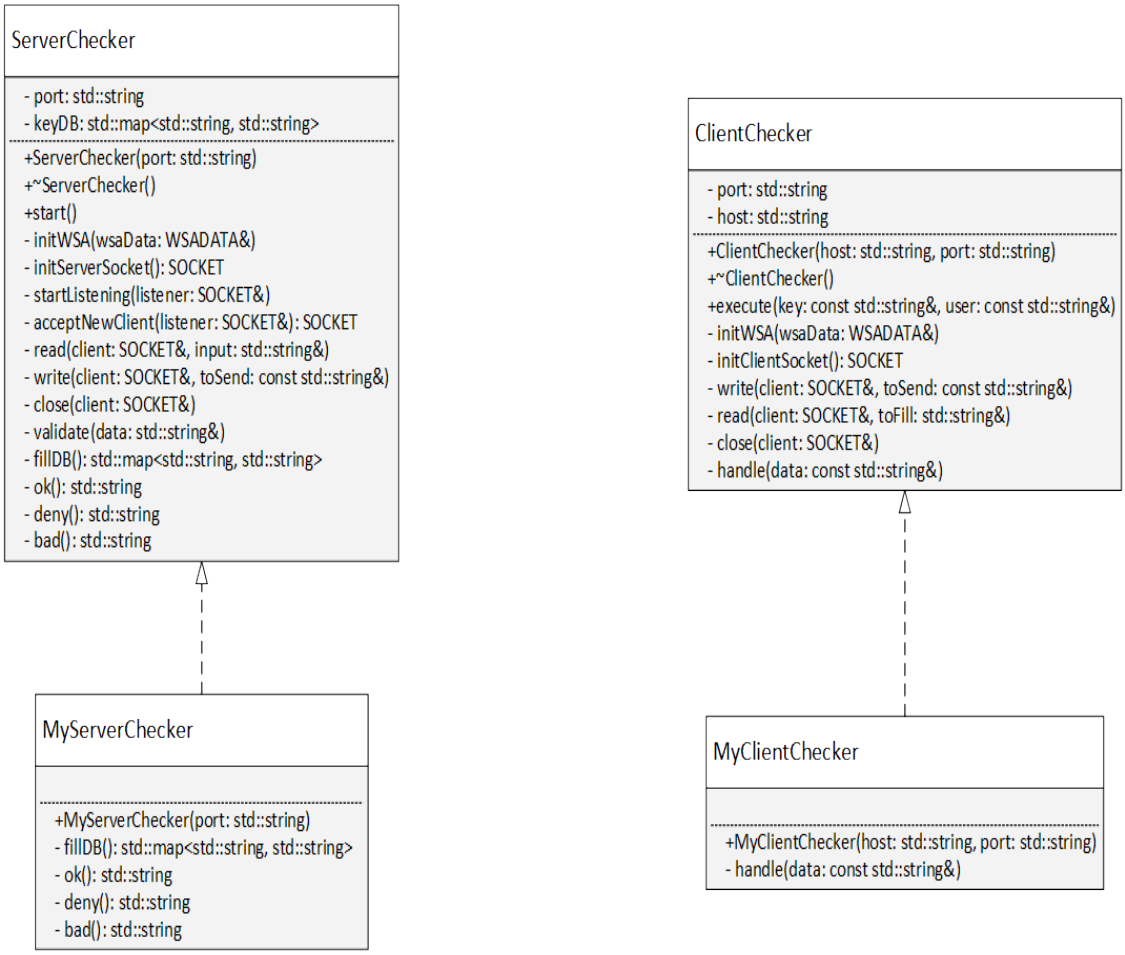


Рисунок 2 - Диаграмма классов

1.3 Разработка диаграмм последовательностей и деятельности

Диаграмма последовательностей отображает взаимодействие объектов в динамике.

Диаграмма последовательностей относится к диаграммам взаимодействия UML, описывающим поведенческие аспекты системы, но рассматривает взаимодействие объектов во времени. Другими словами, диаграмма последовательностей отображает временные особенности передачи и приема сообщений объектами.

Данные диаграммы содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста.

Объекты обозначаются прямоугольниками с подчеркнутыми именами, сообщения - линиями со стрелками, возвращаемые результаты - пунктирными линиями со стрелками. Прямоугольники на вертикальных линиях под каждым из объектов показывают "время жизни" объектов. Впрочем, довольно часто их не изображают на диаграмме, все это зависит

от индивидуального стиля проектирования. [3]

Диаграмма последовательностей приведена на рисунке 3.

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности. Диаграмма деятельности - это своеобразная блок-схема, которая описывает последовательность выполнения операций во времени. Их можно использовать для моделирования динамических аспектов поведения системы. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии.

В диаграммах деятельности используются пиктограммы "действие", "переход", "выбор" и "линии синхронизации". В языке UML действие изображается в виде прямоугольника с закругленными углами, переходы - в виде направленных стрелок, элементы выбора - в виде ромбов, линии синхронизации - в виде горизонтальных и вертикальных линий.

Состояние действия является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока используются переходы, показывающие путь из одного состояния действия или деятельности в другое.

Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования любого потока управления. Как и в блок-схеме, вы можете включить в модель выбор, который описывает различные пути выполнения в зависимости от значения некоторого булевского выражения.

Простые и ветвящиеся последовательные переходы в диаграммах деятельности используются чаще всего. Однако можно встретить и параллельные потоки, и это особенно характерно для моделирования бизнес-процессов. В UML для обозначения разделения и слияния таких параллельных потоков выполнения используются линии синхронизации, которые рисуется в виде жирной вертикальной или горизонтальной линии.[3]

Диаграмма последовательностей приведена на рисунке 4.

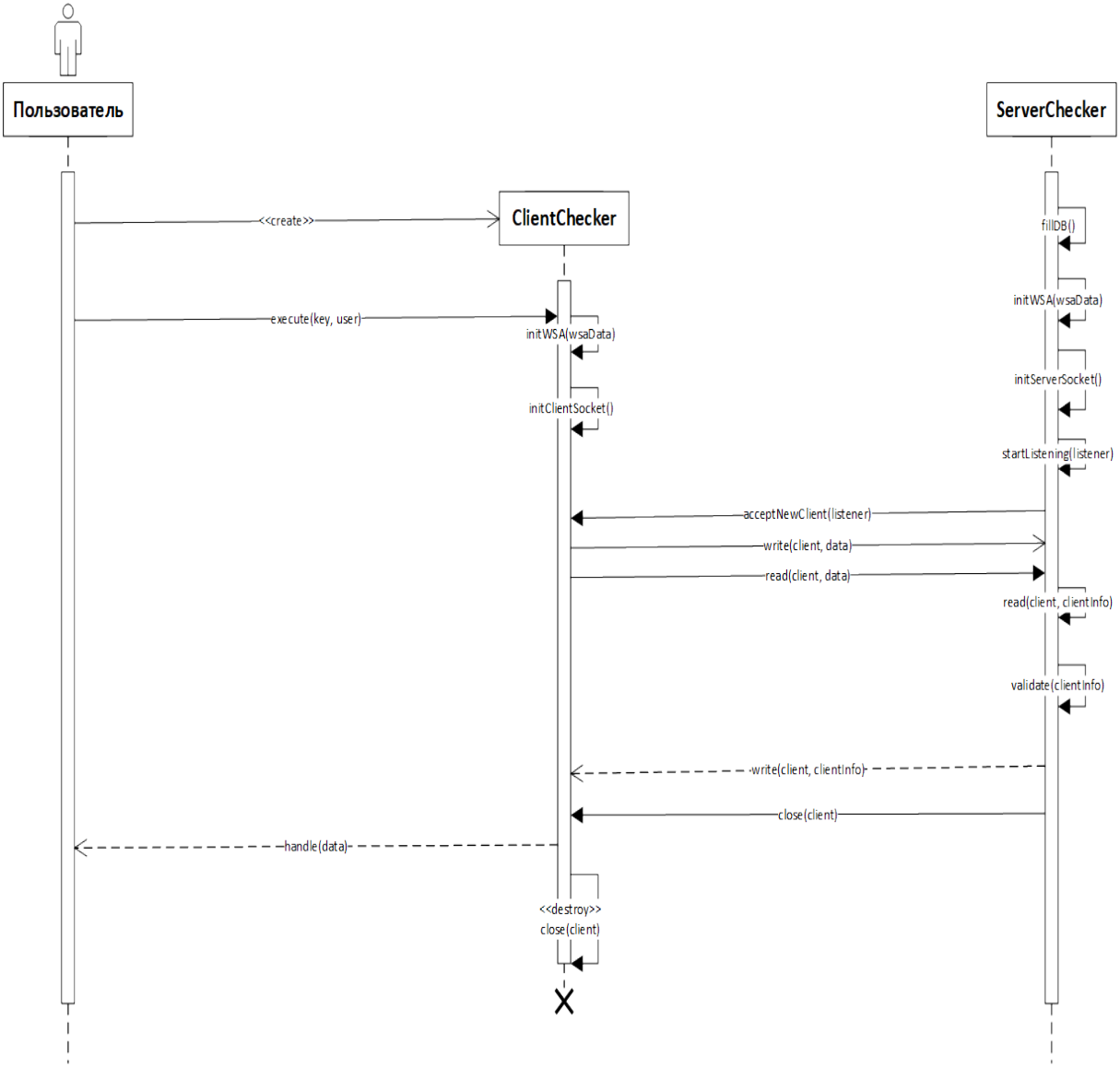


Рисунок 3 - Диаграмма последовательностей

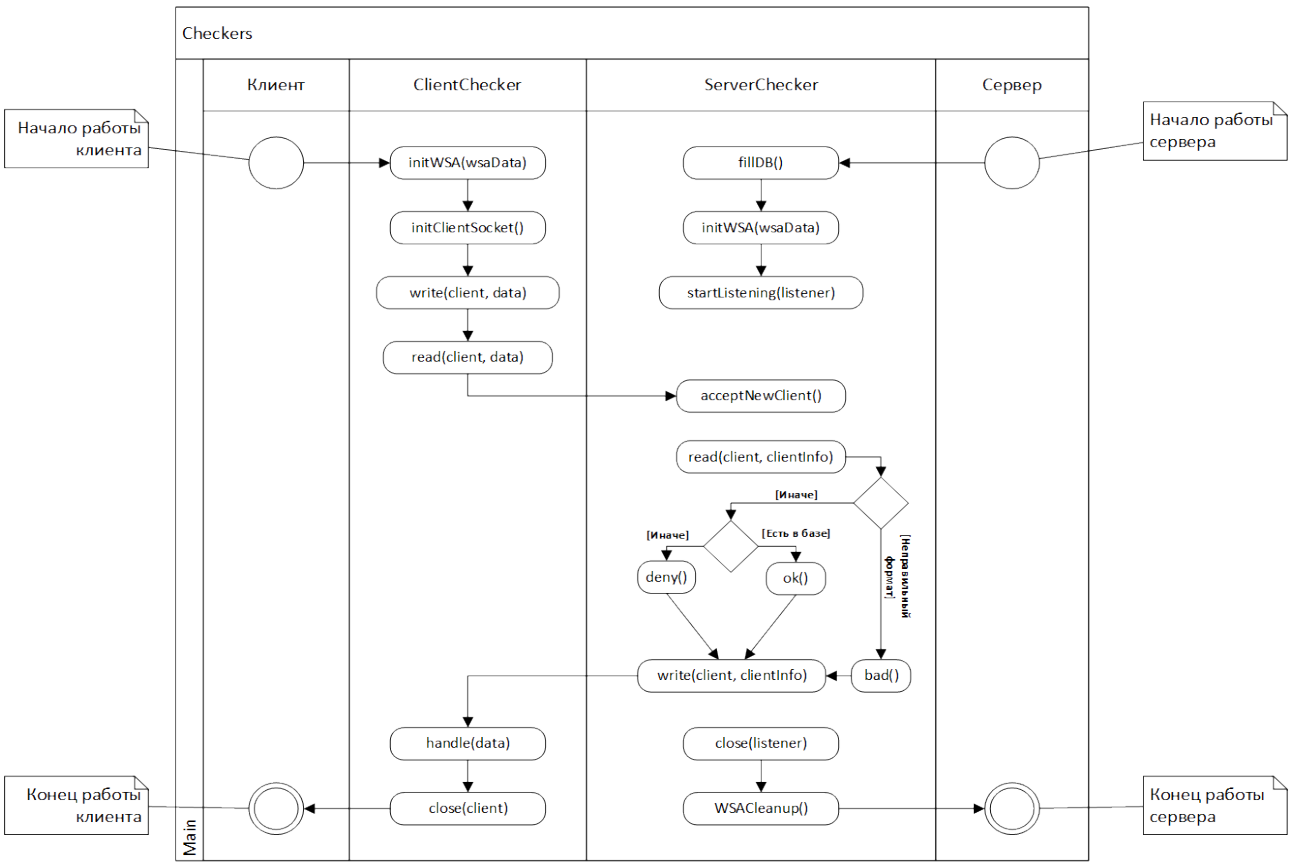


Рисунок 4 - Диаграмма деятельности

1.4 Разработка программного модуля

Модули проверки лицензии программы в режиме онлайн были разработаны в соответствии со всеми требованиями, представленными в техническом задании. Модули полностью соответствуют словесному описанию, представленному в пункте 1.1. Код заголовочного модуля представлен в приложении А. Код реализации класса представлен в приложении Б.

Для демонстрации работы модулей была разработана демонстрационная программа. Данная программа принимает на вход данные пользователя и лицензионный ключ, вводимые с клавиатуры и обрабатывает их на сервере. Выходным параметром программы является сообщение о результате проверки ключа на подлинность. Код демонстрационной программы представлен в приложении В.

Пример запуска программы представлен на рисунке 5.

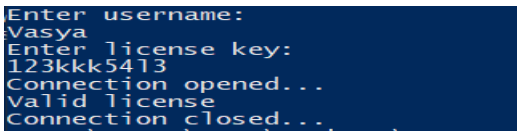


Рисунок 5 - Пример работы модуля

1.5 Разработка модульных тестов

Для проверки работоспособности модулей были разработаны модульные тесты. Результат представлен в таблице 1.

Таблица 1 - Модульные тесты

Тест	Входные данные	Ожидаемый результат
Тест на вводимое имя пользователя	Vasya 123kkk54l3	Valid license
	Vas1li 123kkk54l3	Invalid license
	V@sya 123kkk54l3	Invalid license
	Vas111i 123kkk54l3	Invalid license
Тест на вводимый ключ	Vasya 123kkk54l3	Valid license
	Vasya l2344413	Invalid license
	Vasya 124bgавfds3	Invalid request

1.6 Разработка функциональных тестов

На данном этапе был разработан тест в связи с тем что не имеет смысла тестировать работу сервера на его стороне, были разработаны тесты для клиентского приложения. Сервер развёрнут на localhost и использует порт 8080.

Таблица 2 -Тестирование объекта MyClientChecker.

Входные параметры	Ожидаемый результат
MyClientChecker("localhost", 8080);	Программы отработала успешно
MyClientChecker("localhost", 80);	Socket connect failed
MyClientChecker("192.168.0.1", 8080);	Программа отработала успешно
MyClientChecker("133.145.122.2", 1234);	Socket connect failed

1.7 Документирование программы с использованием Doxygen.

На данном этапе была выполнена разработка документации с использованием Doxygen. Doxygen — это кроссплатформенная система документирования исходных текстов, которая поддерживает C++, Си, Objective-C, Python, Java, IDL, PHP, C#, Фортран, VHDL и, частично, D. Doxygen генерирует документацию на основе набора исходных текстов и также может быть настроен для извлечения структуры программы из недокументированных исходных кодов. Возможно составление графов зависимостей программных объектов, диаграмм классов и исходных кодов с гиперссылками. При создании документации были использованы команды:

@brief используется для указания краткого описания

/// однострочный блок.

\code и \endcode добавление кода к документации для иллюстрации работы.

** и */ многострочный блок в JavaDoc стиль.

Конфигурационный файл представляет собой набор конфигурационных переменных и их описание . В конфигурационный файл был внесен ряд изменений:

PROJECT_NAME был установлен на Система проверки лицензии программы онлайн.

OUTPUT_LANGUAGE был установлен на Russian.

EXTRACT ALL был установлен на YES

EXTRACT_PRIVATE был установлен на YES

2 Экспериментальная часть

2.1 Обнаружение дефектов в модуле с помощью модульного тестирования

Модуль был протестирован на предмет выявления дефектов. Тесты были произведены на основании модульных и функциональных тестов. Результат представлен в таблице 3.

Таблица 3 - Результаты тестирования

Тест	Входные данные	Ожидаемый результат	Полученный результат	Результат тестирования
Тест на вводимое имя пользователя	Vasya 123kkk54l3	Valid license	Valid license	Тест пройден
	Vas1li 123kkk54l3	Invalid license	Invalid license	Тест пройден
	V@sya 123kkk54l3	Invalid license	Invalid license	Тест пройден
	Vas1l1i 123kkk54l3	Invalid license	Invalid license	Тест пройден
Тест на вводимый ключ	Vasya 123kkk54l3	Valid license	Valid license	Тест пройден
	Vasya l2344413	Invalid license	Invalid license	Тест пройден
	Vasya 124bgaвfds3	Invalid request	Invalid request	Тест пройден
Тест объекта MyClientChecker	MyClientChecker("localhost", 8080);	Программы отработала успешно	Программы отработала успешно	Тест пройден
	MyClientChecker("localhost", 80)	Socket connect failed	Socket connect failed	Тест пройден
	MyClientChecker("192.168.0.1", 8080);	Программы отработала успешно	Программы отработала успешно	Тест пройден
	MyClientChecker("133.145.122.2", 1234);	Socket connect failed	Socket connect failed	Тест пройден

2.2 Проверка модуля на соответствие техническому заданию с помощью квалифицированного тестирования.

Для проведения квалифицированного тестирования было использовано юнит-тестирование с использованием библиотеки UnitTest++.

Была написана программа для проведения тестирования. Код программы приведен в приложении Г.

Заключение

В рамках настоящей курсовой работы был предложен алгоритм реализации модуля проверки лицензионного ключа программы в режиме онлайн.

На первом этапе было составлено словесное описание модуля.

На втором этапе была разработана диаграммы вариантов использования и классов.

На третьем этапе были составлены диаграммы последовательностей и деятельности.

На четвертом этапе был разработан модуль проверки лицензии в режиме онлайн.

На пятом этапе была произведена разработка модульных тестов.

На шестом этапе была произведена разработка функциональных тестов.

На седьмом этапе было проведено обнаружение дефектов в модуле с помощью тестирования.

На восьмом этапе была составлена документация программы с использованием Doxygen.

Все поставленные перед разработчиком задачи были выполнены в полном объеме.

Список используемых источников

1. Лицензирование программного обеспечения// habr.com. - 2016. URL: https://habr.com/ru/post/275995/

2. Winsock URL: https://docs.microsoft.com/en-us/windows/desktop/winsock/getting-started-with-winsock

3. UML диаграммы URL: http://book.uml3.ru/sec_1_5

4. Документирование URL:http://www.doxygen.nl/manual/

Приложение А

(обязательное)

Код заголовочного модуля

Файл Includes.h

```
#pragma once

#pragma comment(lib, "Ws2_32.lib")
#include <iostream>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <cstdio>
#include <cstdlib>
#include <string>
#include <map>
#include <regex>
```

Файл ServerChecker.h

```
#pragma once

#include "includes.h"

class ServerChecker
{
private:
	//Порт сервера
	std::string port;
	//Словарь с лицензионными ключами и пользователями
	std::map<std::string, std::string> keyDB;
	void initWSA(WSADATA& wsaData);
	SOCKET initServerSocket();
	void startListening(SOCKET& listener);
	SOCKET acceptNewClient(SOCKET& listener);
	void read(SOCKET& client, std::string& input);
	void write(SOCKET& client, const std::string& toSend);
	void close(SOCKET& client);
	void validate(std::string& data);
	//Функции, которую надо определить при наследовании
	virtual std::map<std::string, std::string> fillDB() = 0;
	virtual std::string ok() = 0;
	virtual std::string deny() = 0;
	virtual std::string bad() = 0;
public:
	ServerChecker(std::string port);
	~ServerChecker() = default;
	void start();
};
```

Файл ClientChecker.h

```
#pragma once
```

```
#include "includes.h"

class ClientChecker
{
private:
	//Адрес сервера
	std::string host;
	//Порт сервера
	std::string port;
	void initWSA(WSADATA& wsaData);
	SOCKET initClientSocket();
	void write(SOCKET& client, const std::string& toSend);
	void read(SOCKET& client, std::string& toFill);
	//Функция, которую надо определить при наследовании
	virtual void handle(const std::string& toHandle) = 0;
	void close(SOCKET& client);
public:
	ClientChecker(std::string host, std::string port);
	~ClientChecker() = default;
	void execute(const std::string& key, const std::string& user);
};
```

Приложение Б

(обязательное)

Код реализации классов.

Файл ClientChecker.cpp

```
#include "ClientChecker.h"


//Размер буфера
const int bufferSize = 512;

//Инициализация Socket API
void ClientChecker::initWSA(WSADATA& wsaData)
{
	int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (iResult != 0) {
		perror("WSA Startup failed: ");
		exit(1);
	}
}

//Создание подключения
SOCKET ClientChecker::initClientSocket()
{
	struct addrinfo* result = nullptr, info;
	ZeroMemory(&info, sizeof(info));
	//Выбор протокола
	info.ai_family = AF_UNSPEC;
	//Выбор типа подключения
	info.ai_socktype = SOCK_STREAM;
	//Используется TCP/IP протокол
	info.ai_protocol = IPPROTO_TCP;

	//Получение данных о подключении
	int iResult = getaddrinfo(host.c_str(), port.c_str(), &info, &result);
	if (iResult != 0) {
		perror("Getting of addrinfo failed: ");
		WSACleanup();
		exit(2);
	}

	SOCKET toReturn = INVALID_SOCKET;

	//Перебираем возможные подключения
	for (auto ptr = result; ptr != nullptr; ptr = ptr->ai_next) {
		//Пробуем создать подключение
		toReturn = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
		if (toReturn == INVALID_SOCKET) {
			perror("Socket init failed: ");
			WSACleanup();
```

```
                exit(3);
            }

            //Пробуем связаться с сервером
            iResult = connect(toReturn, ptr->ai_addr, (int)ptr->ai_addrlen);
            if (iResult == SOCKET_ERROR) {
                closesocket(toReturn);
                toReturn = INVALID_SOCKET;
                continue;
            }
            break;
        }

        if (toReturn == INVALID_SOCKET) {
            perror("Socket connect failed: ");
            WSACleanup();
            exit(4);
        }

        freeaddrinfo(result);
        puts("Connection opened...");
        fflush(stdout);
        return toReturn;
}

//Отправка строки в установленное подключение
void ClientChecker::write(SOCKET& client, const std::string& toSend)
{
        int iResult = send(client, toSend.c_str(), toSend.size(), 0);
        if (iResult == SOCKET_ERROR) {
            perror("Sending failed: ");
            closesocket(client);
            WSACleanup();
            exit(5);
        }
}

//Получение ответа от сервера
void ClientChecker::read(SOCKET& client, std::string& toFill)
{
        int iResult;
        //Инициализация буфера
        char buf[bufferSize + 1];
        toFill.clear();
        do {
            ZeroMemory(buf, sizeof(buf));
            //Получаем информацию
            iResult = recv(client, buf, bufferSize, 0);
            //Если она получена
            if (iResult > 0) {
```

```
                toFill += buf;
            }
            else {
                perror("Receiving failed: ");
                closesocket(client);
                WSACleanup();
                exit(6);
            }
            //Пока информация осталась
        } while (iResult == bufferSize);
}

//Обрываем подключение
void ClientChecker::close(SOCKET & client)
{
        puts("Connection closed...");
        //Закрываем подключение
        closesocket(client);
        //Отключаем Socket API
        WSACleanup();
        fflush(stdout);
}

//Конструктор
ClientChecker::ClientChecker(std::string host, std::string port)
        : host(host), port(port) {}

//Метод запуска запроса
void ClientChecker::execute(const std::string & key, const std::string & user)
{
        WSADATA wsaData;
        initWSA(wsaData);

        SOCKET client = initClientSocket();

        //Создание JSON объекта с данными
        std::string data = "{ \"key\": \"";
        data += key;
        data += "\", \"user\": \"";
        data += user;
        data += "\" }";

        write(client, data);
        read(client, data);
        //Управление ответом
        handle(data);
        close(client);
        }
```

Файл ServerChecker.cpp

```
	#include "ServerChecker.h"

//Размер буфера
const int bufferSize = 512;

//Конструктор
ServerChecker::ServerChecker(std::string port)
	: port(port) {}

//Метод запуска сервера
void ServerChecker::start()
{
	//Инициализация словаря
	keyDB = fillDB();

	WSADATA wsaData;
	initWSA(wsaData);

	SOCKET listener = initServerSocket();

	startListening(listener);

	//Непрерывно принимаем запросы
	while (true) {
		SOCKET client = acceptNewClient(listener);
		std::string clientInfo;
		read(client, clientInfo);
		validate(clientInfo);
		write(client, clientInfo);
		close(client);
	}

	//Закрываем порт
	closesocket(listener);
	//Отключаем Socket API
	WSACleanup();
}

//Инициализация Socket API
void ServerChecker::initWSA(WSADATA& wsaData)
{
	int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (iResult != 0) {
		perror("WSA Startup failed: ");
		exit(1);
	}
}

//Открытие серверного порта
SOCKET ServerChecker::initServerSocket()
```

```
{
	struct addrinfo* result = nullptr, info;
	ZeroMemory(&info, sizeof(info));
	//Устанавливаем IPv4
	info.ai_family = AF_INET;
	//Выбор типа подключения
	info.ai_socktype = SOCK_STREAM;
	//Используем TCP/IP протокол
	info.ai_protocol = IPPROTO_TCP;
	//Автоопределение
	info.ai_flags = AI_PASSIVE;

	//Получение данных о серверном порте
	int iResult = getaddrinfo(nullptr, port.c_str(), &info, &result);
	if (iResult != 0) {
		perror("Getting of standart addrinfo failed: ");
		WSACleanup();
		exit(2);
	}

	SOCKET toReturn = INVALID_SOCKET;
	//Инициализация серверного порта
	toReturn = socket(result->ai_family, result->ai_socktype, result->ai_protocol);

	if (toReturn == INVALID_SOCKET) {
		perror("Server socket init failed: ");
		WSACleanup();
		exit(3);
	}

	//Привязка порта к адресу
	iResult = bind(toReturn, result->ai_addr, (int)result->ai_addrlen);
	if (iResult == SOCKET_ERROR) {
		perror("Binding failed: ");
		freeaddrinfo(result);
		closesocket(toReturn);
		WSACleanup();
		exit(4);
	}

	freeaddrinfo(result);
	puts("Server started...");
	fflush(stdout);
	return toReturn;
}

//Начинаем слушать запросы на порт
void ServerChecker::startListening(SOCKET& listener)
{
	if (listen(listener, SOMAXCONN) == SOCKET_ERROR) {
```

```
		perror("Listening failed: ");
		closesocket(listener);
		WSACleanup();
		exit(5);
	}
}

//Устанавливаем подключение с клиентом
SOCKET ServerChecker::acceptNewClient(SOCKET& listener)
{
	SOCKET toReturn = INVALID_SOCKET;

	toReturn = accept(listener, nullptr, nullptr);
	if (toReturn == INVALID_SOCKET) {
		perror("Accepting failed: ");
		closesocket(listener);
		WSACleanup();
		exit(6);
	}
	puts("Connection opened...");
	fflush(stdout);
	return toReturn;
}

//Считываем содержимое запроса
void ServerChecker::read(SOCKET& client, std::string& input)
{
	int iResult;
	//Инициализация буфера
	char buf[bufferSize + 1];
	do {
		ZeroMemory(buf, sizeof(buf));
		//Получаем информацию
		iResult = recv(client, buf, bufferSize, 0);
		//Если она получена
		if (iResult > 0) {
			input += buf;
		}
		else {
			perror("Receiving failed: ");
			closesocket(client);
			WSACleanup();
			exit(7);
		}
		//Пока информация осталась
	} while (iResult == bufferSize);
}

//Отправка ответа
void ServerChecker::write(SOCKET & client, const std::string & toSend)
```

```
{
	int iResult = send(client, toSend.c_str(), toSend.size(), 0);
	if (iResult == SOCKET_ERROR) {
		perror("Sending failed: ");
		closesocket(client);
		WSACleanup();
		exit(8);
	}
}

//Обрываем подключение клиента
void ServerChecker::close(SOCKET & client)
{
	puts("Connection closing...");
	//Закрываем подключение
	int iResult = shutdown(client, SD_SEND);
	if (iResult == SOCKET_ERROR) {
		perror("Shutdown failed: ");
		closesocket(client);
		WSACleanup();
		exit(8);
	}
	//Удаляем информацию о нём
	closesocket(client);
	fflush(stdout);
}

//Проводим валидацию данных
void ServerChecker::validate(std::string & data)
{
	//Лог
	printf("Login attempt with: %s\n", data.c_str());
	fflush(stdout);
	//Объект с содержимым от регулярного выражения
	std::smatch matches;
	//Если JSON корректен
	if (std::regex_match(data, matches,
		std::regex(R"(\s*\{\s*"key"\s*:\s*"(\w+)\"\s*,\s*"user"\s*:\s*"(\w+)\"\s*\}\s*)"))) {
		//Получаем содержимое ключа
		std::string key = matches[1];
		//Получаем содержимое имени пользователя
		std::string user = matches[2];
		//Если существует лицензионный ключ и он принадлежит пользователю
		if (keyDB.count(key) && keyDB[key] == user) {
			//Корректный ответ
			data = ok();
		}
		else {
			//Некорректный ответ
			data = deny();
```

```
            }
        }
        //То же самое, только поля идут наоборот
        else if (std::regex_match(data, matches,
            std::regex(R"(\s*\{\s*"user"\s*:\s*"(\w+)\"\s*,\s*"key"\s*:\s*"(\w+)\"\s*\}\s*)"))) {
            std::string key = matches[2];
            std::string user = matches[1];
            if (keyDB.count(key) && keyDB[key] == user) {
                data = ok();
            }
            else {
                data = deny();
            }
        }
        //Некорректный запрос
        else {
            data = bad();
        }
        fflush(stdout);
}
```

Приложение В

(обязательное)

Код демонстрационной программы.

Файл CourseClient.cpp

```
	#include "ClientChecker.h"

std::string result;
//Наследуем наш класс от библиотечного
class MyClientChecker : public ClientChecker {
private:
	//Переопределяем управление ответом
	void handle(const std::string& toHandle) {
		//Выводим информацию в основной поток
		result = toHandle;
		puts(toHandle.c_str());
		fflush(stdout);
	}
public:
	//Переопределяем конструктор
	MyClientChecker(std::string host, std::string port)
		: ClientChecker(host, port) {}
};

//В аргументах надо передать адрес сервера и порт если он не равен 80
int main(int argc, char** argv) {
	if (argc == 1 || argc > 3) {
		printf("Usage: %s host [port=80]\n", argv[0]);
		return 0;
	}
	std::string host = argv[1];
	std::string port = argc == 3 ? argv[2] : "80";

	//Создаём объект
	MyClientChecker checker(host, port);

	std::string user, key;
	std::cout << "Enter username: " << std::endl;
	//Считываем имя пользователя
	std::getline(std::cin, user);
	std::cout << "Enter license key: " << std::endl;
	//Считываем лицензионный ключ
	std::getline(std::cin, key);
	//Отправляем запрос на сервер
	checker.execute(key, user);
	return 0;
	}
```

Файл CourseServer.cpp

```
	#include "ServerChecker.h"
```

```
//Наследуем наш класс от библиотечного
class MyServerChecker : public ServerChecker {
private:
	//Переопределяем метод заполнения словаря с лицензионными ключами
	std::map<std::string, std::string> fillDB() {
		std::map<std::string, std::string> toReturn;
		toReturn.insert({ "123kkk54l3", "Vasya" });
		return toReturn;
	}
	//Переопределяем сообщение при найденном совпадении
	std::string ok() {
		return "Valid license";
	}
	//Переопределяем сообщение при некорректном запросе
	std::string bad() {
		return "Invalid request";
	}
	//Переопределяем сообщение при отсутствии совпадения
	std::string deny() {
		return "Invalid license";
	}
public:
	//Переопределяем конструктор
	MyServerChecker(std::string port)
		: ServerChecker(port) {}
};

//В аргументах надо передать порт если он не равен 80
int main(int argc, char** argv) {
	if (argc == 1) {
		//Создаём на стандартном порту объект сервера
		MyServerChecker checker("80");
		//Запуск
		checker.start();
	}
	else if (argc == 2) {
		//Создаём на определённом порту объект сервера
		MyServerChecker checker((std::string)argv[1]);
		//Запуск
		checker.start();
	}
	else {
		printf("Usage: %s [port=80]\n", argv[0]);
	}
	return 0;
	}
```

Приложение Г

(обязательное)

Код юнит-тестирования

```
#include <UnitTest++/UnitTest++.h>
#include "CourseClient.cpp"
TEST(UserName)
{
	MyClientChecker("localhost", 80).execute("Vasya", "123kkk54l3");
	CHECK_EQUAL(result, "Valid license");
	MyClientChecker("localhost", 80).execute("Vas1li", "123kkk54l3");
	CHECK_EQUAL(result, "Invalid license");
	MyClientChecker("localhost", 80).execute("V@sya", "123kkk54l3");
	CHECK_EQUAL(result, "Invalid license");
	MyClientChecker("localhost", 80).execute("Vas111i", "123kkk54l3");
	CHECK_EQUAL(result, "Invalid license");
}
TEST(TestKey)
{
	MyClientChecker("localhost", 80).execute("Vasya", "123kkk54l3");
	CHECK_EQUAL(result, "Valid license");
	MyClientChecker("localhost", 80).execute("Vasya", "l2344413");
	CHECK_EQUAL(result, "Invalid license");
	MyClientChecker("localhost", 80).execute("Vasya", "124bgавfds3");
	CHECK_EQUAL(result, "Invalid request");
}
TEST(Object)
{
	MyClientChecker("localhost", 8080).execute("Vasya", "123kkk54l3");
	CHECK_THROW(MyClientChecker("localhost", 80).execute("Vasya", "123kkk54l3"), int)
	MyClientChecker("192.168.0.1", 8080).execute("Vasya", "123kkk54l3");
	CHECK_THROW(MyClientChecker("133.145.122.2",1234).execute("Vasya",
"123kkk54l3"), int)
}
int main(int argc, char** argv)
{
	return UnitTest::RunAllTests();
	}
```

Приложение Д

(обязательное)

Документирование программы с использованием Doxygen.

Система проверки лицензии программы онлайн

Создано системой Doxygen 1.8.15

1 Иерархический список классов 1
1.1 Иерархия классов . . . 1

2 Алфавитный указатель классов 3
2.1 Классы . . . 3

3 Список файлов 5
3.1 Файлы . . . 5

4 Классы 7
4.1 Класс ClientChecker . . . 7
4.1.1 Конструктор(ы) . . . 8
4.1.1.1 ClientChecker() . . . 8
4.1.1.2 ~ClientChecker() . . . 8
4.1.2 Методы . . . 8
4.1.2.1 close() . . . 8
4.1.2.2 execute() . . . 9
4.1.2.3 handle() . . . 9
4.1.2.4 initClientSocket() . . . 9
4.1.2.5 initWSA() . . . 10
4.1.2.6 read() . . . 10
4.1.2.7 write() . . . 11
4.1.3 Данные класса . . . 11
4.1.3.1 host . . . 11
4.1.3.2 port . . . 11
4.2 Класс MyClientChecker . . . 11
4.2.1 Подробное описание . . . 12
4.2.2 Конструктор(ы) . . . 12
4.2.2.1 MyClientChecker() . . . 12
4.2.3 Методы . . . 12
4.2.3.1 handle() . . . 12
4.3 Класс MyServerChecker . . . 13
4.3.1 Подробное описание . . . 13
4.3.2 Конструктор(ы) . . . 13
4.3.2.1 MyServerChecker() . . . 13
4.3.3 Методы . . . 14
4.3.3.1 bad() . . . 14
4.3.3.2 deny() . . . 14
4.3.3.3 fillDB() . . . 14
4.4 Класс ServerChecker . . . 15
4.4.1 Конструктор(ы) . . . 16
4.4.1.1 ServerChecker() . . . 16
4.4.1.2 ~ServerChecker() . . . 16
4.4.2 Методы . . . 16

4.4.2.1 acceptNewClient() . . . 16
4.4.2.2 bad() . . . 16
4.4.2.3 close() . . . 17
4.4.2.4 deny() . . . 17
4.4.2.5 fillDB() . . . 17
4.4.2.6 initServerSocket() . . . 18
4.4.2.7 initWSA() . . . 18
4.4.2.8 ok() . . . 19
4.4.2.9 read() . . . 19
4.4.2.10 start() . . . 19
4.4.2.11 startListening() . . . 20
4.4.2.12 validate() . . . 20
4.4.2.13 write() . . . 21
4.4.3 Данные класса . . . 21
4.4.3.1 keyDB . . . 21
4.4.3.2 port . . . 21

5 Файлы 23
5.1 Файл ClientChecker.cpp . . . 23
5.1.1 Переменные . . . 23
5.1.1.1 bufferSize . . . 23
5.2 Файл ClientChecker.h . . . 23
5.3 Файл CourseClient.cpp . . . 24
5.3.1 Функции . . . 24
5.3.1.1 main() . . . 24
5.4 Файл CourseServer.cpp . . . 24
5.4.1 Функции . . . 25
5.4.1.1 main() . . . 25
5.5 Файл includes.h . . . 25
5.6 Файл ServerChecker.cpp . . . 25
5.6.1 Переменные . . . 26
5.6.1.1 bufferSize . . . 26
5.7 Файл ServerChecker.h . . . 26

Предметный указатель 27

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

ClientChecker 7
MyClientChecker 11
ServerChecker 15
MyServerChecker 13

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

ClientChecker 7
MyClientChecker
Наследуем наш класс от библиотечного 11
MyServerChecker
Наследуем наш класс от библиотечного 13
ServerChecker 15

Глава 3

Список файлов

3.1 Файлы

Полный список файлов.

ClientChecker.cpp 23
ClientChecker.h 23
CourseClient.cpp 24
CourseServer.cpp 24
includes.h 25
ServerChecker.cpp 25
ServerChecker.h 26

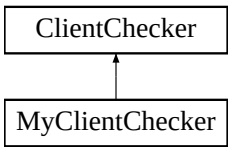
Глава 4

Классы

4.1 Класс ClientChecker

#include <ClientChecker.h>

Граф наследования:ClientChecker:



Открытые члены

- ClientChecker (std::string host, std::string port)
  Конструктор
- ~ClientChecker ()=default
- void execute (const std::string &key, const std::string &user)
  Метод запуска запроса

Закрытые члены

- void initWSA (WSADATA &wsaData)
  Инициализация Socket API.
- SOCKET initClientSocket ()
  Создание подключения
- void write (SOCKET &client, const std::string &toSend)
  Отправка строки в установленное подключение
- void read (SOCKET &client, std::string &toFill)
  Получение ответа от сервера
- virtual void handle (const std::string &toHandle)=0
  Функция, которую надо определить при наследовании
- void close (SOCKET &client)
  Обрываем подключение

Закрытые данные

- std::string host
  Адрес сервера
- std::string port
  Порт сервера

4.1.1 Конструктор(ы)

4.1.1.1 ClientChecker()

```
ClientChecker::ClientChecker (
                std::string host,
                std::string port )
```

Конструктор

4.1.1.2 ~ClientChecker()

```
ClientChecker::~ClientChecker ( )  [default]
```

4.1.2 Методы

4.1.2.1 close()

```
void ClientChecker::close (
                SOCKET & client )  [private]
```

Обрываем подключение
```
void ClientChecker::close(SOCKET & client)
{
    puts("Connection closed...
);
```

Закрываем подключение
```
closesocket(client);
```

Отключаем Socket API
```
WSACleanup();
fflush(stdout);
```

4.1.2.2 execute()

void ClientChecker::execute (
const std::string & key,
const std::string & user)

Метод запуска запроса
```
void ClientChecker::execute(const std::string & key, const std::string & user)
{
    WSADATA wsaData;
    initWSA(wsaData);
    SOCKET client = initClientSocket();
```

Создание JSON объекта с данными
```
std::string data = "{ \"key\": \;
data += key;
data += "\ \"user\": \;
data += user;
data += "\" }
;
write(client, data);
read(client, data);
```

Управление ответом
```
handle(data);
close(client);
```

4.1.2.3 handle()

virtual void ClientChecker::handle (
const std::string & toHandle) [private], [pure virtual]

Функция, которую надо определить при наследовании

Замещается в MyClientChecker (стр. 12).

4.1.2.4 initClientSocket()

SOCKET ClientChecker::initClientSocket ( ) [private]

Создание подключения
```
SOCKET ClientChecker::initClientSocket()
{
    struct addrinfo* result = nullptr, info;
    ZeroMemory(&info, sizeof(info));
```

Выбор протокола
```
info.ai_family = AF_UNSPEC;
```

Выбор типа подключения
```
info.ai_socktype = SOCK_STREAM;
```

Используется TCP/IP протокол
```
info.ai_protocol = IPPROTO_TCP;
```

Получение данных о подключении
```
int iResult = getaddrinfo(host.c_str(), port.c_str(), &info, &result);
if (iResult != 0) {
    perror("Getting of addrinfo failed:
);
    WSACleanup();
    exit(2);
}
```

Перебираем возможные подключения
```
for (auto ptr = result; ptr != nullptr; ptr = ptr->ai_next) {
```

@brief Пробуем создать подключение
\code
toReturn = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
if (toReturn == INVALID_SOCKET) {
    perror("Socket init failed: ");
    WSACleanup();
    exit(3);
}

@brief Пробуем связаться с сервером
\code
iResult = connect(toReturn, ptr->ai_addr, (int)ptr->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    closesocket(toReturn);
    toReturn = INVALID_SOCKET;
    continue;
}
break;

}

4.1.2.5 initWSA()

void ClientChecker::initWSA (
WSADATA & wsaData) [private]

Инициализация Socket API.
{
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        perror("WSA Startup failed:
);
        exit(1);
    }
}

4.1.2.6 read()

void ClientChecker::read (
SOCKET & client,
std::string & toFill) [private]

Получение ответа от сервера
void ClientChecker::read(SOCKET& client, std::string& toFill)
{
    int iResult;

Инициализация буфера
char buf[bufferSize + 1];
toFill.clear();
do {
    ZeroMemory(buf, sizeof(buf));

Получаем информацию
iResult = recv(client, buf, bufferSize, 0);

Если она получена
if (iResult > 0) {
    toFill += buf;
}
else {
    perror("Receiving failed:
);
    closesocket(client);
    WSACleanup();
    exit(6);
}

    @brief Пока информация осталась
    \code
} while (iResult == bufferSize);

}

4.1.2.7 write()

```
void ClientChecker::write (
                SOCKET & client,
                const std::string & toSend )   [private]
```

Отправка строки в установленное подключение
```
void ClientChecker::write(SOCKET& client, const std::string& toSend)
{
    int iResult = send(client, toSend.c_str(), toSend.size(), 0);
    if (iResult == SOCKET_ERROR) {
        perror("Sending failed:
");
        closesocket(client);
        WSACleanup();
        exit(5);
    }
}
```

4.1.3 Данные класса

4.1.3.1 host

```
std::string ClientChecker::host   [private]
```

Адрес сервера
```
std::string host;
```

4.1.3.2 port

```
std::string ClientChecker::port   [private]
```

Порт сервера
```
std::string port;
```

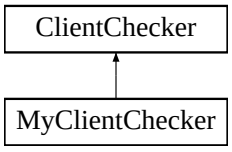
Объявления и описания членов классов находятся в файлах:

- ClientChecker.h
- ClientChecker.cpp

4.2 Класс MyClientChecker

Наследуем наш класс от библиотечного

Граф наследования:MyClientChecker:



Открытые члены

- MyClientChecker (std::string host, std::string port)
  Переопределяем конструктор

Закрытые члены

- void handle (const std::string &toHandle)
  Переопределяем управление ответом

4.2.1 Подробное описание

Наследуем наш класс от библиотечного

4.2.2 Конструктор(ы)

4.2.2.1 MyClientChecker()

```
MyClientChecker::MyClientChecker (
            std::string host,
            std::string port )   [inline]
```

Переопределяем конструктор

4.2.3 Методы

4.2.3.1 handle()

```
void MyClientChecker::handle (
            const std::string & toHandle )   [inline], [private], [virtual]
```

Переопределяем управление ответом
```
void handle(const std::string& toHandle)
```

Выводим информацию в основной поток
```
puts(toHandle.c_str());
fflush(stdout);
```

Замещает ClientChecker (стр. 9).

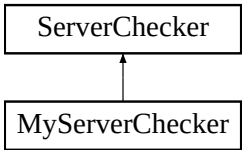
Объявления и описания членов класса находятся в файле:

- CourseClient.cpp

4.3 Класс MyServerChecker

Наследуем наш класс от библиотечного

Граф наследования:MyServerChecker:



Открытые члены

- MyServerChecker (std::string port)

Закрытые члены

- std::map< std::string, std::string > fillDB ()

  Переопределяем метод заполнения словаря с лицензионными ключами
- std::string bad ()

  Переопределяем сообщение при найденном совпадении
- std::string deny ()

  Переопределяем сообщение при отсутствии совпадения

4.3.1 Подробное описание

Наследуем наш класс от библиотечного

4.3.2 Конструктор(ы)

4.3.2.1 MyServerChecker()

```
MyServerChecker::MyServerChecker (
                std::string port )   [inline]
```

@brief Переопределяем конструктор
\code
MyServerChecker(std::string port)
: ServerChecker(port) {}

};

4.3.3 Методы

4.3.3.1 bad()

std::string MyServerChecker::bad ( ) [inline], [private], [virtual]

Переопределяем сообщение при найденном совпадении
```
std::string ok() {
    return "Valid license
;
}
```

std::string ok() (стр. 18) { return "Valid license"; } /** Переопределяем сообщение при некорректном запросе
```
std::string bad() {
    return "Invalid request
;
}
```

Замещает ServerChecker (стр. 16).

4.3.3.2 deny()

std::string MyServerChecker::deny ( ) [inline], [private], [virtual]

Переопределяем сообщение при отсутствии совпадения
```
std::string deny() {
    return "Invalid license
;
}
```

Замещает ServerChecker (стр. 17).

4.3.3.3 fillDB()

std::map<std::string, std::string> MyServerChecker::fillDB ( ) [inline], [private], [virtual]

Переопределяем метод заполнения словаря с лицензионными ключами

Замещает ServerChecker (стр. 17).

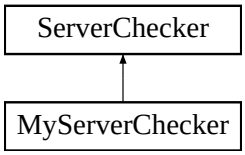
Объявления и описания членов класса находятся в файле:

- CourseServer.cpp

4.4 Класс ServerChecker

#include <ServerChecker.h>

Граф наследования:ServerChecker:



Открытые члены

- ServerChecker (std::string port)
  Конструктор
- ~ServerChecker ()=default
- void start ()
  Метод запуска сервера

Закрытые члены

- void initWSA (WSADATA &wsaData)
  Инициализация Socket API.
- SOCKET initServerSocket ()
  Открытие серверного порта
- void startListening (SOCKET &listener)
  Начинаем слушать запросы на порт
- SOCKET acceptNewClient (SOCKET &listener)
  Устанавливаем подключение с клиентом
- void read (SOCKET &client, std::string &input)
  Считываем содержимое запроса
- void write (SOCKET &client, const std::string &toSend)
  Отправка ответа
- void close (SOCKET &client)
  Обрываем подключение клиента
- void validate (std::string &data)
  Проводим валидацию данных
- virtual std::map< std::string, std::string > fillDB ()=0
  Функции, которую надо определить при наследовании
- virtual std::string ok ()=0
- virtual std::string deny ()=0
- virtual std::string bad ()=0

Закрытые данные

- std::string port
  Порт сервера
- std::map< std::string, std::string > keyDB
  Словарь с лицензионными ключами и пользователями

4.4.1 Конструктор(ы)

4.4.1.1 ServerChecker()

```
ServerChecker::ServerChecker (
            std::string port )
```

Конструктор

4.4.1.2 ~ServerChecker()

```
ServerChecker::~ServerChecker ( )   [default]
```

4.4.2 Методы

4.4.2.1 acceptNewClient()

```
SOCKET ServerChecker::acceptNewClient (
            SOCKET & listener )   [private]
```

Устанавливаем подключение с клиентом

4.4.2.2 bad()

```
virtual std::string ServerChecker::bad ( )   [private], [pure virtual]
```

Замещается в MyServerChecker (стр. 14).

4.4.2.3 close()

void ServerChecker::close (
SOCKET & client) [private]

Обрываем подключение клиента
void ServerChecker::close(SOCKET & client)
{
puts("Connection closing...
);

Закрываем подключение
int iResult = shutdown(client, SD_SEND);
if (iResult == SOCKET_ERROR) {
perror("Shutdown failed:
);
closesocket(client);
WSACleanup();
exit(8);
}

@brief Удаляем информацию о нём
\code
closesocket(client);
fflush(stdout);

}

4.4.2.4 deny()

virtual std::string ServerChecker::deny ( ) [private], [pure virtual]

Замещается в MyServerChecker (стр. 14).

4.4.2.5 fillDB()

virtual std::map<std::string, std::string> ServerChecker::fillDB ( ) [private], [pure virtual]

Функции, которую надо определить при наследовании

Замещается в MyServerChecker (стр. 14).

4.4.2.6 initServerSocket()

SOCKET ServerChecker::initServerSocket ( ) [private]

Открытие серверного порта
SOCKET ServerChecker::initServerSocket()
{
struct addrinfo* result = nullptr, info;
ZeroMemory(&info, sizeof(info));

Устанавливаем IPv4
info.ai_family = AF_INET;

Выбор типа подключения
info.ai_socktype = SOCK_STREAM;

Используем TCP/IP протокол
info.ai_protocol = IPPROTO_TCP;

Автоопределение
info.ai_flags = AI_PASSIVE;

Получение данных о серверном порте int iResult = getaddrinfo(nullptr, port.c_str(), &info, &result); if (iResult != 0) { perror("Getting of standart addrinfo failed: "); WSACleanup(); exit(2); }

SOCKET toReturn = INVALID_SOCKET;

Инициализация серверного порта
toReturn = socket(result->ai_family, result->ai_socktype, result->ai_protocol);
if (toReturn == INVALID_SOCKET) {
perror("Server socket init failed:
");
WSACleanup();
exit(3);
}

@brief Привязка порта к адресу
\codeiResult = bind(toReturn, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
perror("Binding failed: ");
freeaddrinfo(result);
closesocket(toReturn);
WSACleanup();
exit(4);
}

freeaddrinfo(result);
puts("Server started...");
fflush(stdout);
return toReturn;

}

4.4.2.7 initWSA()

void ServerChecker::initWSA (
WSADATA & wsaData) [private]

Инициализация Socket API.

4.4.2.8 ok()

virtual std::string ServerChecker::ok ( ) [private], [pure virtual]

4.4.2.9 read()

void ServerChecker::read (
SOCKET & client,
std::string & input) [private]

Считываем содержимое запроса
```
void ServerChecker::read(SOCKET& client, std::string& input)
{
    int iResult;
```

Инициализация буфера
```
char buf[bufferSize + 1];
do {
    ZeroMemory(buf, sizeof(buf));
```

Получаем информацию
```
iResult = recv(client, buf, bufferSize, 0);
```

Если она получена
```
if (iResult > 0) {
    input += buf;
}
else {
    perror("Receiving failed:
");
    closesocket(client);
    WSACleanup();
    exit(7);
}
```

Пока информация осталась
```
while (iResult == bufferSize);
```

4.4.2.10 start()

void ServerChecker::start ( )

Метод запуска сервера
```
void ServerChecker::start()
```

Инициализация словаря
```
keyDB = fillDB();
WSADATA wsaData;
initWSA(wsaData);
SOCKET listener = initServerSocket();
startListening(listener);
```

Непрерывно принимаем запросы
```
while (true) {
    SOCKET client = acceptNewClient(listener);
    std::string clientInfo;
    read(client, clientInfo);
    validate(clientInfo);
    write(client, clientInfo);
    close(client);
}
```

Закрываем порт
```
closesocket(listener);
```

Отключаем Socket API
```
WSACleanup();
```

4.4.2.11 startListening()

void ServerChecker::startListening (
SOCKET & listener) [private]

Начинаем слушать запросы на порт

4.4.2.12 validate()

void ServerChecker::validate (
std::string & data) [private]

Проводим валидацию данных
void ServerChecker::validate(std::string & data)

Лог
```
printf("Login attempt with: %s\n
, data.c_str());
fflush(stdout);
```

Объект с содержимым от регулярного выражения
```
std::smatch matches;
```

Если JSON корректен
```
if (std::regex_match(data, matches,
    std::regex(R"(\s*\{\s*"key"\s*:\s*"(\w+)\"\s*,\s*"user"\s*:\s*"(\w+)\"\s*\}\s*)")))
```

Получаем содержимое ключа
```
std::string key = matches[1];
```

Получаем содержимое имени пользователя
```
std::string user = matches[2];
```

Если существует лицензионный ключ и он принадлежит пользователю
```
if (keyDB.count(key) && keyDB[key] == user) {
```

Корректный ответ
```
data = ok();
```

Некорректный ответ
```
data = deny()
```

То же самое, только поля идут наоборот
```
else if (std::regex_match(data, matches,
    std::regex(R"(\s*\{\s*"user"\s*:\s*"(\w+)\"\s*,\s*"key"\s*:\s*"(\w+)\"\s*\}\s*)"))) {
    std::string key = matches[2];
    std::string user = matches[1];
    if (keyDB.count(key) && keyDB[key] == user) {
        data = ok();
    }
    else {
        data = deny();
    }
}
```

Некорректный запрос
```
else {
    data = bad();
}
fflush(stdout);
```

4.4.2.13 write()

void ServerChecker::write (
SOCKET & client,
const std::string & toSend) [private]

Отправка ответа

4.4.3 Данные класса

4.4.3.1 keyDB

std::map<std::string, std::string> ServerChecker::keyDB [private]

Словарь с лицензионными ключами и пользователями

4.4.3.2 port

std::string ServerChecker::port [private]

Порт сервера

Объявления и описания членов классов находятся в файлах:

- ServerChecker.h
- ServerChecker.cpp

Глава 5

Файлы

5.1 Файл ClientChecker.cpp

#include "ClientChecker.h"

Переменные

- const int bufferSize = 512
  Размер буфера

5.1.1 Переменные

5.1.1.1 bufferSize

const int bufferSize = 512

Размер буфера

5.2 Файл ClientChecker.h

#include "includes.h"

Классы

- class ClientChecker

5.3 Файл CourseClient.cpp

#include "ClientChecker.h"

Классы

- class MyClientChecker

  Наследуем наш класс от библиотечного

Функции

- int main (int argc, char **argv)

5.3.1 Функции

5.3.1.1 main()

```
int main (
                int argc,
                char ** argv )
```

"brief В аргументах надо передать адрес сервера и порт если он не равен 80
```
int main(int argc, char** argv) {
    if (argc == 1 || argc > 3) {
        printf("Usage: %s host [port=80]\n
, argv[0]);
        return 0;
    }
    std::string host = argv[1];
    std::string port = argc == 3 ? argv[2] : "80
;
```

Создаём объект
```
MyClientChecker checker(host, port);
std::string user, key;
```

Считываем имя пользователя
```
std::getline(std::cin, user);
std::cout « "Enter license key:
 « std::endl;
```

Считываем лицензионный ключ
```
std::getline(std::cin, key);
```

Отправляем запрос на сервер
```
checker.execute(key, user);
return 0;
```

5.4 Файл CourseServer.cpp

#include "ServerChecker.h"

Классы

- class MyServerChecker

  Наследуем наш класс от библиотечного

Функции

- int main (int argc, char **argv)

  В аргументах надо передать порт если он не равен 80.

5.4.1 Функции

5.4.1.1 main()

```
int main (
                int argc,
                char ** argv )
```

В аргументах надо передать порт если он не равен 80.

```
int main(int argc, char** argv) {
    if (argc == 1)
```

Создаём на стандартном порту объект сервера

```
MyServerChecker checker("80
");
```

Запуск

```
checker.start();
```

Создаём на определённом порту объект сервера

```
MyServerChecker checker((std::string)argv[1]);
```

Запуск

```
checker.start();
```

5.5 Файл includes.h

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <string>
#include <map>
#include <regex>
```

5.6 Файл ServerChecker.cpp

```
#include "ServerChecker.h"
```

Переменные

- const int bufferSize = 512
  Размер буфера

5.6.1 Переменные

5.6.1.1 bufferSize

const int bufferSize = 512

Размер буфера

5.7 Файл ServerChecker.h

#include "includes.h"

Классы

- class ServerChecker

Предметный указатель

~ClientChecker
    ClientChecker, 8
~ServerChecker
    ServerChecker, 16

acceptNewClient
    ServerChecker, 16

bad
    MyServerChecker, 14
    ServerChecker, 16
bufferSize
    ClientChecker.cpp, 23
    ServerChecker.cpp, 26

ClientChecker, 7
    ~ClientChecker, 8
    ClientChecker, 8
    close, 8
    execute, 8
    handle, 9
    host, 11
    initClientSocket, 9
    initWSA, 10
    port, 11
    read, 10
    write, 10
ClientChecker.cpp, 23
    bufferSize, 23
ClientChecker.h, 23
close
    ClientChecker, 8
    ServerChecker, 16
CourseClient.cpp, 24
    main, 24
CourseServer.cpp, 24
    main, 25

deny
    MyServerChecker, 14
    ServerChecker, 17

execute
    ClientChecker, 8

fillDB
    MyServerChecker, 14
    ServerChecker, 17

handle
    ClientChecker, 9
    MyClientChecker, 12
host
    ClientChecker, 11

includes.h, 25
initClientSocket
    ClientChecker, 9
initServerSocket
    ServerChecker, 17
initWSA
    ClientChecker, 10
    ServerChecker, 18

keyDB
    ServerChecker, 21

main
    CourseClient.cpp, 24
    CourseServer.cpp, 25
MyClientChecker, 11
    handle, 12
    MyClientChecker, 12
MyServerChecker, 13
    bad, 14
    deny, 14
    fillDB, 14
    MyServerChecker, 13

ok
    ServerChecker, 18

port
    ClientChecker, 11
    ServerChecker, 21

read
    ClientChecker, 10
    ServerChecker, 19

ServerChecker, 15
    ~ServerChecker, 16
    acceptNewClient, 16
    bad, 16
    close, 16
    deny, 17
    fillDB, 17
    initServerSocket, 17
    initWSA, 18
    keyDB, 21

ok, 18
port, 21
read, 19
ServerChecker, 16
start, 19
startListening, 19
validate, 20
write, 20
ServerChecker.cpp, 25
bufferSize, 26
ServerChecker.h, 26
start
ServerChecker, 19
startListening
ServerChecker, 19

validate
ServerChecker, 20

write
ClientChecker, 10
ServerChecker, 20